C introduction

Variables

Richard Mörbitz, Manuel Thieme

Contents

Overview

Data types

Identifiers

Variable I/O

Exercise

Everything's a number

In C, the concept of variables has been adopted from mathematics.

- Used to remember values during program execution
- Stored in the memory
- Accessed by an identifier
- In memory, all variables are numbers (sequences of bits)
- Computer interpretes them as different data types

Useage

Declaration:

```
type identifier;
```

Assignment:

```
identifier = value;
```

Definition (all at once):

```
type identifier = value;
```

Example:

```
int number;                     /* declaration */
number = 42;                    /* assignment */
int another_number = 23;        /* definition */
```

Saving lines

Multiple declarations

```
int number, another_number;
```

Multiple Definitions

```
int number = 42, anothernumber = 23;
```

But be careful:

```
int a = 23, b = 23;
```

$\neq$

```
int a, b = 23;
```

Integer numbers

- Keywords: *int*, *short*, *long*
- Stored as a binary number with fixed length
- Can be *signed*(default) or *unsigned*
- Actual size of *int*, *short*, *long* depends on architecture

Example (64 Bit):

```
int a;              /* Range: -2.147.483.648 to 2.147.483.647 */
unsigned short b;                          /* Range: 0 to 65.535 */
```

Floating point numbers

- Keywords: *float*, *double*, *long double*
- Stored as specified in *IEEE 754 Standard* TL;DR
- Special values for ∞, $-\infty$, NaN
- Useful for fractions and very large numbers
- Type a decimal point instead of a comma! (Since C was originally developed in the USA)

Example:

```
float x = 0.125;              /* Precision: 7 to 8 digits */
double y = 111111.111111;     /* Precision: 15 to 16 digits */
```

Characters

- Keyword: *char*
- Can be *signed*(default) or *unsigned*
- Size: 1 Byte (8 Bit) on almost every architechture
- Indented to represent a single character
- Stores its *ASCII* number (e.g. 'A' $\Rightarrow$ 65)

You can define a *char* either by its ASCII number or by its symbol:

```
char a = 65;
char b = 'A';    /* use single quotation marks */
```

Valid identifiers

- Consist of English letters (no *ß*, *ä*, *ö*, *ü*), numbers and underscore (_)
- Start with a letter or underscore
- Are case sensitive (*number* differs from *Number*)
- Are unique - must not be redeclared
- Must not be reserved words (e.g *int*, *return*)

Speaking identifiers

```c
/* calculate volume of square pyramid */
int a, b, c;
a = 3;
b = 2;
c = (1 / 3) * a * a * b;
```

```c
/* calculate volume of square pyramid */
int length, height, volume;
length = 3;
height = 2;
volume = (1 / 3) * length * length * height;
```

Use speaking identifiers.

Please, use speaking identifiers.[1]

[1] Seriously, use speaking identifiers.

Scopes

- Program area in which an identifier may be used
- Referring to it anywhere else causes compilation errors
- Starts at the line of declaration
- Ends at the end of the block, in which the variable was declared

You begin a block with a '{' and end it with a '}':

```
1  int main(int argc, char *argv[]) {
2      int i = 4;
3      {
4          i = 3;          /* valid */
5          int j = 5;
6      }
7      j = 2;              /* invalid, j is not in scope */
8
9      return 0;
10 }
```

Overwriting identifiers

When redeclaring identifiers inside a block, they refer to a new variable:

```
1 int main(char argc, char *argv[]) {
2     int i = 3;          /* Defining i with value 3 */
3     {
4         i = 2;          /* "Outside" i is now 2 */
5         int i = 4;      /* New "inside" i */
6         i = 7;          /* Changes "inside" i only */
7     }
8                         /* "Outside" i is still 2 */
9 }
```

Style:

- Stay in one language (English recommended)
- Decide wheter to use *camelCaseIdentifiers* or *underscore_identifiers*.
- When nesting blocks, indent every inner block by one additional tab!

printf() with placeholders

The string you pass to *printf* can contain placeholders:

```
int a = 3, b = 5;
float c = 7.4;
printf("a: %d\n", a);
printf("b: %d\nc: %f\n", b, c);
```

Output:

```
a: 3
b: 5
c: 7.4
```

You can insert any amount of placeholders. For each placeholder, you have to pass a value of the corresponding type.

Example placeholders

The placeholder determines how the value is interpreted. To avoid compiler warnings, only use the following combinations:

type	**description**	**type of argument**
%c	single character	char, int (if <= 255)
%d	decimal number	char, int
%u	unsigned decimal number	unsigned char, unsigned int
%X	hexadecimal number	char, int
%ld	long decimal number	long
%f	floating point number	float, double

Variable input

scanf() is another useful function from the standard library.

- Like *printf()*, it is declared in *stdio.h*
- Like *printf()*, it has a format string with placeholders
- You can use it to read values of primitive datatypes from the command line

Example:

```
int i;
scanf("%d", &i);
```

After calling *scanf()*, the program waits for the user to input a value in the command line. After pressing the *return* key, that value is stored in *i*.

Note:

- *scanf()* uses the same placeholders as *printf()*
- You must type an *&* before each variable identifier (more about this later)
- If you read a number (using %d, %u etc.), interpretation
  - Starts at first digit
  - Ends before last non digit character
- If you use %c, the first character of the user input is interpreted (this may be a ' ' as well!)

Never trust the user: they may enter a blank line while you expect a number, which means your input variable is still undefined!

Hello again!

- Write a program that prints the String "Hello World!" on the command line, using only placeholders in the format string.
- **Experts:** In leetspeek you say "H3110 W0|21d!". Use actual numbers.

ASCII number explorer

- Write a program that asks to input a character and prints its ASCII number
- **Experts**: write a Program that asks the user for 5 characters and prints them in reversed order.

ASCII number explorer

- Write a program that asks to input a character and prints its ASCII number
  - Hint: it is a matter of interpretation
- **Experts**: write a Program that asks the user for 5 characters and prints them in reversed order.